# Soulmate Audit Report

Version 1.0

*Securigor.io*

February 13, 2024

# Soulmate Audit Report

Securigor.io

February 12, 2024

Prepared by: Securigor Lead Auditors:

- bigBagBoogy

## Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
  - Roles
  - Issues found
- Findings
  - High
    * [H-1] The ERC721.sol::_safeMint functions The protocol uses have their `require` statements after the executing code.
    * [H-2] Only every second soulmate gets an NFT
  - Medium
    * [M-1] In Staking.sol::claimRewards the user gets "cheated" out of staked time by `lastClaim[msg.sender] = block.timestamp;`.
    * [M-2] In Staking.sol, the functions `deposit`, `claimRewards` and `withdraw` all do not follow CEI. The emits of the events are after external transfers have been done.

- Low
  - [L-1] Indirect circular dependency bug
  - [L-2] A user can get divorced even if not in a couple.
- Gas
  - [gas] Redundant writing to storage

## Protocol Summary

Audit of the Soulmate protocol, where you can mint your shared Soulbound NFT with an unknown person, and get LoveToken as a reward for staying with your soulmate. A staking contract is available to collect more love. Because if you give love, you receive more love.

## Disclaimer

The Securigor team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

## Risk Classification

| | | Impact | | |
|---|---|---|---|---|
| | | High | Medium | Low |
| | High | H | H/M | M |
| Likelihood | Medium | H/M | M | M/L |
| | Low | M | M/L | L |

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

## Audit Details

- Commit Hash: main branch ## Scope
- In Scope: Entire Protocol

### Roles

None

### Issues found

| Severity | Number of issues found |
|---|---|
| High | 1 |
| Medium | 2 |
| Low | 2 |
| Gas | 1 |
| Total | 6 |

## Findings

### High

#### [H-1] The ERC721.sol::_safeMint functions The protocol uses have their `require` statements after the executing code.

**Description:** In the `ERC721.sol` contract, the `_safeMint` functions have their require statements placed after the execution of the _mint function. This means that the minting operation occurs before checking whether the recipient address is a smart contract or not. As a result, tokens can be minted to contracts (when to.code.length > 0), potentially leading to unexpected behavior or security vulnerabilities.

**Impact:** This coding pattern could result in tokens being mistakenly transferred to contracts that are not intended to receive ERC721 tokens. If the recipient contract does not handle ERC721 tokens correctly, it could result in loss of tokens or unexpected behavior.

**Proof of Concept:**

```
contract MaliciousContract {
    function onERC721Received(address _operator, address _from, uint256
        _tokenId, bytes calldata _data) external returns(bytes4) {
        // Malicious behavior here
        return bytes4(keccak256("onERC721Received(address,address,
           uint256,bytes)"));
    }
}
```

**Recommended Mitigation:** The `require` statements should be placed before executing the _mint function to ensure that tokens are only minted to addresses that are not smart contracts or that properly handle ERC721 tokens. Here's the corrected version of the `_safeMint` function. Please follow this pattern for both `_safeMint` functions.

```
function _safeMint(address to, uint256 id) internal virtual {
    require(
        to.code.length == 0 || ERC721TokenReceiver(to).onERC721Received
           (msg.sender, address(0), id, "") == ERC721TokenReceiver.
           onERC721Received.selector,
        "UNSAFE_RECIPIENT"
    );

    _mint(to, id);
}
```

### [H-2] Only every second soulmate gets an NFT

**Description:** In the README is says: "The Soulbound NFT ***shared*** by soulmates used in the protocol" However, the only thing soulmates share is an ID in the `idToOwners` mapping. Only every second soulmate actually gets an NFT. ***It's like: "Let's buy a rembrand together 50-50, but we'll hang it at my house, forever"*** To add insult to injury, the `idToOwners` mapping is marked **`private`**, so there's not even an (easy) way to view or show your "indirect" ownership.

**Impact:** If owning the NFT is going to be a very coveted thing at some point, There might even grow a thing where users will try to "time" their entry into the "marriage" to make sure they are the second entrant in a pair.

**Proof of Concept:** Please paste this test at the bottom in `SoulmateTest.t.sol` and run: `forge test --mt test_onlySoulmate2GetsNFT -vvvvv`

```
 function test_onlySoulmate2GetsNFT() public {
        vm.prank(soulmate1);
        soulmateContract.mintSoulmateToken();
        vm.prank(soulmate2);
```

```
            soulmateContract.mintSoulmateToken();
            soulmateContract.ownerOf(0); // soulmate2
        }
```

***stack-trace:*** `emit Transfer(from: 0x0000000000000000000000000000000000000000, to: soulmate2: [0xe93A5E9F20AF38E00a08b9109D20dEc1b965E891], id: 0)`

**Recommended Mitigation:** One option is to mint 2 NFT's with cross-refrencing meta-data per couple and send each soulmate one. Or, wile the README claims: "It is used by Airdrop.sol and Staking.sol to know how long the couple are in love." Since this is only half-true at best, the protocol could choose to completely get rid of the NFT's and just keep the mapping of id to soulmates [2].

## Medium

### [M-1] In Staking.sol::claimRewards the user gets "cheated" out of staked time by `lastClaim[msg.sender] = block.timestamp;`.

**Description:** When a user claims staking rewards just shy of 2 weeks, she'll get 1 token and has the surplus time (over 1 week) reset to "0"

**Impact:** The impact of the issue is that users claiming staking rewards just shy of two weeks (or any amount of weeks) will receive only one token, effectively losing the surplus time (over one week) as it resets to "0". This results in a loss of potential rewards for users who stake for periods just under two weeks, which may lead to dissatisfaction among users and impact the overall fairness and attractiveness of the staking mechanism.

**Proof of Concept:** Please paste this test at the bottom in `StakingTest.t.sol` and run: `forge test --mt test_claimingAfter13daysBurns6daysOfStaking -vvvvv`

```
    function test_claimingAfter13daysBurns6daysOfStaking() public {
        _depositTokenToStake(1 ether);
        uint256 almost2weeks = 2 weeks - 1 seconds;
        console2.log("almost2weeks: ", almost2weeks); // 1209599
        vm.warp(almost2weeks); // 1209599

        vm.startPrank(soulmate1);
        console2.log("last claim: ", stakingContract.lastClaim(
            soulmate1));
        stakingContract.claimRewards();
        console2.log("soulmate1's lovetoken balance: ", loveToken.
            balanceOf(soulmate1)); // 1
        console2.log("last claim: ", stakingContract.lastClaim(
            soulmate1));

        vm.warp(almost2weeks + 6 days); // now the user is staking for
            20+ days
        vm.expectRevert();
        stakingContract.claimRewards(); // no, even after 20+ days the
            user can't claim a second token, because she minted at the "
            wrong" moment.
        console2.log("soulmate1's lovetoken balance: ", loveToken.
            balanceOf(soulmate1)); // 1
        console2.log("last claim: ", stakingContract.lastClaim(
            soulmate1));
    }
```

**Recommended Mitigation:** Update last claim time by adding 1 week for each week staked

```
+   lastClaim[msg.sender] += 1 weeks * timeInWeeksSinceLastClaim;
-   lastClaim[msg.sender] = block.timestamp;
```

### [M-2] In Staking.sol, the functions `deposit`, `claimRewards` and `withdraw` all do not follow CEI. The emits of the events are after external transfers have been done.

**Description:** The events in the `Staking.sol` contract are emitted after external transfers have been executed in the functions deposit, withdraw, and claimRewards. This violates the Checks-Effects-Interactions pattern, a best practice in Solidity smart contract development. According to this pattern, external calls (interactions) should be made after state changes (effects) to prevent reentrancy attacks and ensure that contract state is consistent before interacting with other contracts or external entities.

**Impact:** Without a dApp, this protocol is no more than a couple of contracts. So inevitably there will have to be an off-chain (website) that will be listening and ***acting*** to events being emitted by the smart-contracts.

**Proof of Concept:** consider a scenario where a malicious contract exploits the lack of proper ordering in the deposit function. By calling the deposit function from a contract that performs a reentrancy attack, it could potentially manipulate the contract state or funds before the Deposited event is emitted, leading to unintended consequences or loss of funds for legitimate users. ***Patrick Collins*** himself validates this finding not one week ago: https://www.linkedin.com/posts/patrickalphac_a-lot-of-people-think-its-fine-to-emit-events-activity-7160679494452715524-Ou3M?utm_source=share&utm_medium=member_desktop

**Recommended Mitigation:** emit events before executing external transfers or interactions.

### Low

#### [L-1] Indirect circular dependency bug

**Description:** The `LoveToken.sol` constructor takes `airdropVault` and `stakingVault` as arguments. They are initialized in `Airdrop.sol`, which, needs an implementation of the `ILoveToken` interface.

**Impact:** For deployment to mainnet or a testnet other than a local environment, the circular dependancies will prevent deployment of `LoveToken.sol`.

**Proof of Concept:** The initialization of `Vault.sol's` vaults (airdropVault and stakingVault) relies on `LoveToken.sol` being deployed and providing an implementation of the `ILoveToken` interface. Therefore, `LoveToken.sol` needs to be deployed before Vault.sol can successfully initialize its vaults.

**Recommended Mitigation:** refactoring the contracts to remove circular dependencies or redesigning the deployment process to handle dependencies more efficiently.

#### [L-2] A user can get divorced even if not in a couple.

**Description:** The Soulmate.sol::getDivorced function does not check if the caller is in a couple. This means that if a "single" soulmate calls `getDivorced`, she will be marked as `divorced`. (from address `0` to be precise)

**Impact:** A user can "shoot herself in the foot", perhaps unintentional. Since this is irreversible, it can be quite annoying.

**Proof of Concept:** Please paste this test at the bottom in `SoulmateTest.t.sol` and run: `forge test --mt test_singleSoulmateCanGetDivorced -vvvvv`

```
function test_singleSoulmateCanGetDivorced() public {
        vm.startPrank(soulmate1);
        soulmateContract.getDivorced();
        console2.log("divorce status is: ", soulmateContract.isDivorced
            ());
        assertEq(soulmateContract.isDivorced(), true);
    }
```

**Recommended Mitigation:** add a require to check if a user is in a couple:

```
    function getDivorced() public {
+       require(soulmateOf[msg.sender] != address(0), "you are not in a
    couple");
        address soulmate2 = soulmateOf[msg.sender];
        divorced[msg.sender] = true;
        divorced[soulmateOf[msg.sender]] = true;
        emit CoupleHasDivorced(msg.sender, soulmate2);
    }
```

## Gas

### gas Redundant writing to storage

**Description:** in Soulmate.sol::mintSoulmateToken the `soulmateOf` mapping is updated to link 2 soulmates together with the line: `soulmateOf[msg.sender] = soulmate1;`. This effectively links them together and either one can be queried to fint the other. There is no need to do this again the "other way around".

**Impact:**

**Proof of Concept:**

**Recommended Mitigation:**

```
        soulmateOf[msg.sender] = soulmate1;
-       soulmateOf[soulmate1] = msg.sender;
```